Correction avec Xcas du stage algorithmique de l'IREM de Grenoble

Renée De Graeve et Bernard Parisse

17 janvier 2010

Table des matières

1 Introduction

Ce document présente des corrections en `Xcas` du stage d'algorithmique proposé en 2009/10 par l'IREM de Grenoble. Il est distinct du document Solutions de l'IREM de Grenoble, dont les auteurs n'ont pas souhaité intégrer de corrections programmées en `Xcas`. Nous profitons de l'indépendance de ce document pour proposer dans certains cas des algorithmes et programmes que nous jugeons mieux adaptés (utilisant d'autres structures de données, ou des fonctions avec paramètres et valeur de retour au lieu d'avoir à faire des entrées et des sorties fastidieuses, ou en simplifiant certains algorithmes) et pour faire le lien avec des algorithmes classiques. Les thèmes abordés sont les suivants

- l'affectation,
- les listes et les chaînes de caractères avec `Xcas`,
- le maximum et le minimum d'une liste de nombres,
- la factorielle
- les algorithmes de PGCD, dont l'algorithme d'Euclide. Pour l'algorithme (inefficace) utilisant la décomposition en facteurs premiers, on présente le crible d'Eratosthene,
- le crible d'Eratosthene,
- les changements de base, où nous utilisons des chaînes de caractères ou des listes de nombres pour écrire les entiers dans une base différente de 10. On y fait le lien avec l'algorithme de Hörner d'évaluation d'un polynôme.
- la somme de grands entiers, le produit par un chiffre puis le produit de deux grands entiers en utilisant les fonctions précédentes, et en évoquant le lien avec les polynômes.
- le jeu de recherche d'un nombre par une méthode dichotomique, on y fait le lien avec la recherche de 0 d'une fonction continue
- les algorithmes avec sortie graphique sont abordés avec la tortue de `Xcas` mais aussi avec les fonctions géométriques de `Xcas`

On pourra aussi consulter avec profit des documents complémentaires dont

- Le tutoriel algorithmique de programmation Xcas paru dans sesamath, novembre 2009
- Le manuel de programmation de Xcas (dans le menu Aide, Manuels)
- Le tutoriel Débuter en calcul formel (menu Aide)
- les documents de la page pédagogique de Xcas
  `http://www-fourier.ujf-grenoble.fr/~parisse/algo.html`

2 Affectation

2.1 Échange du contenu de 2 mémoires

Pour faire un échange du contenu de deux variables on peut écrire la fonction `echange` qui utilise une variable locale `c` qui sert à stocker la valeur de `a`, puis on met dans `a`, la valeur de `b` et dans `b`, la valeur de `c`. On tape :

```
echange(a,b):={
  local c;
  c:=a;
  a:=b;
```

```
  b:=c;
  retourne a,b;
}:;
```

Puis on tape

`a,b :=echange(a,b)`

Remarque : On peut aussi taper directement :

`a,b :=b,a`

Cela n'est pas identique à la suite d'instructions : `a :=b;b :=a;`

Exercice Comparez :
`a :=5;b :=3;`
`a :=a+b;b :=a-b;`
on obtient alors 8 dans `a` et avec 5 dans `b`
avec :
`a :=3;b :=5;`
`a,b :=a+b,a-b;`
on obtient alors 8 dans `a` et avec 2 dans `b`

Exercice Echanger les contenus de `a` et `b` sans utiliser de variable intermédiaire. Donner une preuve de la méthode utilisée.
Solution :

```
a:=a+b;
b:=a-b;
a:=a-b;
```

La preuve faite par le calcul formel. On tape dans un éditeur de programmes :

```
a:=x;
b:=y;
afficher(a,b);
a:=a+b:;
b:=normal(a-b);
a:=normal(a-b);
afficher(a,b);
```

On valide avec OK ou avec la touche `F9` et on obtient : en bleu les 2 affichages intermédiaires : `x,y`
`y,x`
et comme réponse

`x,y,1,Done,x,y,1`

où (`1` est le résultat d'un affichage et `Done` le résultat de `a :=a+b :;` (car cette instruction se termine par `:;`).

Remarque : cet algorithme n'a aucun intérêt en pratique, on a vu comment échanger le contenu de 2 variables en une ligne avec le langage de haut niveau de Xcas. Avec des langages de bas niveau, il existe en général une instruction d'échange (par exemple `swap` en C++), et c'est aussi le cas en assembleur (par exemple `XCHG` sur architecture x86).

2.2 Permutation circulaire de 3 mémoires

On tape dans un éditeur de programmes :

```
a:=x;
b:=y;
c:=z;
afficher(a,b,c);
d:=a:;
a:=b;
b:=c;
c:=d;
afficher(a,b,c);
```

On valide avec OK ou avec la touche `F9` et on obtient : en bleu les 2 résultats des affichages : `x,y,z` et
`y,z,x` et comme réponse
`x,y,z,1,Done,y,z,x,1` (`1` est le résultat des affichages et `Done` le résultat de `d :=a :;` car cette instruction se termine par `:;`).
ou bien on tape :

```
a:=x;
b:=y;
c:=z;
afficher(a,b,c);
a,b,c:=b,c,a;
afficher(a,b,c);
```

On tape dans un éditeur de programmes :

```
a:=x;
b:=y;
c:=z;
afficher(a,b,c);
d:=c:;
c:=b;
b:=a;
a:=d;
afficher(a,b,c);
```

On valide avec OK ou avec la touche `F9` et on obtient : en bleu les 2 affichages intermédiaires :
`x,y,z`
`z,x,y`
et comme réponse

`x,y,z,1,Done,z,x,y,1`

où `1` est le résultat des affichages et `Done` le résultat de `d :=a :;` (car cette instruction se termine par `:;`). On peut permuter les 3 variables en une seule instruction `a,b,c :=c,a,b;` qu'on peut prouver par :

```
a:=x;
b:=y;
c:=z;
afficher(a,b,c);
a,b,c:=c,a,b;
afficher(a,b,c);
```

3 Les listes et les chaînes de caractères avec Xcas

3.1 Les listes

Une liste de nombres est une suite de nombres séparés par une virgule que l'on met entre des crochets.
Exemple

- `[]` est la liste vide (`dim([])`=0),
- `L :=[0,1,10,16]` est une liste de 4 éléments (`dim(L)`=4)
- le premier élément est `L[0]` et il vaut 0...
- le dernier élément est `L[3]` et il vaut 16.
- Si on tape :`L[2] :=4;` alors `L` est la liste `[0,1,4,16]`.
- `[2 $4]` désigne la la liste `[2,2,2,2]`, `[k$(k=0..3)]` désigne la liste `[0,1,2,3]` et `[k^2$(k=0..4)]` désigne la liste `[0,1,4,9,16]`,
- si `f` est une fonction, `makelist(f,0,3)` désigne la liste `[f(0),f(1),f(2),f(3)]` par exemple `makelist(x->x,0,3)` désigne la liste `[0,1,2,3]` et `makelist(x->x^2,0,4)` désigne la liste `[0,1,4,9,16]`.

On peut taper une fonction `Permutation` qui effectue une permutation sur les éléments d'une liste qui consiste à mettre le premier élément en fin de liste (`tail(L)` est la liste `L` privée de son premier élément) et `append(L,a` met `a` à la fin de la liste `L` alors que `prepend(L,a` met `a` au début de la liste `L`. On tape :

```
Permutation(L):={
  retourne append(tail(L),L[0]);
}:;
```

3.2 Les chaînes de caractères

Une chaîne de caractères est délimitée par deux `"`.
Exemple

- `""` est une chaîne vide (`dim("")`=0)
- `S :="salut"` est une chaîne de 5 caractères (`dim(S)`=5)
- le premier élément est `S[0]` et il vaut le caractère "s"...
- le dernier élément est `S[4]` et il vaut le caractère "t".
- `C :="256"` est une chaîne de 3 caractères (`dim(C)`=3)
- le premier élément est `C[0]` et il vaut le caractère "2"...
- le dernier élément est `C[2]` et il vaut le caractère "6".
- si on tape :`C[2] :="4";`, alors `C` est la chaîne `"254"`.

Pour concaténer 2 chaînes on peut utiliser `+`.
On tape : `B :="Bon"+"jour"`
et `B` est la chaîne "Bonjour".

Lorsqu’on a une chaîne de chiffres on peut la convertir en un nombre avec `expr`. On tape : `expr("136")`. On obtient le nombre 136.
Réciproquement, on peut convertir un nombre en une chaîne de chiffres soit avec `string`, soit avec la concaténation avec la chaîne vide.
On tape : `a :=136;b :=string(a);c :=a+"";d :=""+a`
On obtient la chaîne "136" dans `b`, dans `c` et dans `d`.

4 Maximum et minimum d’une liste de nombres

4.1 Le maximum d’une liste de nombres

On tape :

```
//renvoie le plus grand nombre de L
Max(L):={
  local M,j,d;
  d:=dim(L)-1;
  si d==0 alors
     retourne "liste vide"
  fsi;
  M:=L[0];
  pour j de 1 jusque d faire
    si M< L[j] alors
      M:=L[j];
    fsi;
  fpour;
  retourne M;
}:;
```

4.2 Le maximum et le minimum d’une liste de nombres

On tape :

```
//renvoie le plus grand et le plus petit nombre de L
Maxmin(L):={
  local M,m,j,d;
  d:=dim(L)-1;
  si d==0 alors
     retourne "liste vide"
  fsi;
  M:=L[0];
  m:=L[0];
  pour j de 1 jusque d faire
    si M< L[j] alors
      M:=L[j];
    sinon
      si m> L[j] alors
        m:=L[j];
      fsi;
```

```
    fsi;
  fpour;
  retourne M,m;
}:;
```

4.3 Les deux plus grands nombres d'une liste de nombres

On tape :

```
// renvoie les 2 plus grands nombres de L
// par ordre decroissant
Max2(L):={
 local M1,M2,j,d;
 d:=dim(L)-1;
 si d==0 alors retourne "liste vide" fsi;
 si d==1 alors retourne L[0],L[0];fsi;
 M1:=max(L[0],L[1]);
 M2:=min(L[0],L[1]);
 pour j de 2 jusque d faire
  si M1< L[j] alors M2:=M1;M1:=L[j];
  sinon
   si M2<L[j] alors M2:=L[j]; fsi;
  fsi;
 fpour;
 retourne M1,M2;
}
:;
```

5 Factorielle

5.1 Factorielle et itération

On tape :

```
//fonction qui renvoie n! si n>=0
Factorielle(n):={
  local j,fact;
  fact:=1;
  pour j de 1 jusque n faire
    fact:=fact*j;
  fpour;
  retourne fact;
}:;
```

5.2 Factorielle et récursivité

On tape :

```
Factorieller(n):={
  si n==0 alors retourne 1;fsi;
```

```
  retourne n*Factorieller(n-1);
}:;
```

Exercice Calculer $1 + \frac{1}{1!} + \frac{1}{2!} + ... + \frac{1}{n!}$.
On tape :

```
Sommee(n):={
  local fact, j;
  fact:=1;
  S:=1;
  pour j de 1 jusque n faire
    fact:=fact*j;
    S:=S+1/fact;
  fpour;
  retourne evalf(S),S;
}:;
```

On calcule ici dans la foulée la valeur de $j!$ (on n'utilise pas la fonction précédente pour ne pas ralentir l'exécution du programme). On tape :
`evalf(e),Sommee(20)`
On obtient :
`2.71828182846,`
`2.71828182846,6613313319248080001/2432902008176640000`
On remarque que `Xcas` fait du calcul exact et que `S` est une fraction alors que `evalf(S)` renvoie une valeur approchée de `S`.
Si vous changez le nombre de Digits par exemple :
On tape :
`Digits :=20 :;evalf(e),Sommee(20)`
On obtient :
`"Done",2.71828182845904523535,2.71828182845904523532,`
`6613313319248080001/2432902008176640000`

6 Pgcd

6.1 Pgcd et itération

L'algorithme d'Euclide se sert des deux propriétés :
- si `b==0` alors `pgcd(a,0)=a`
- si `b !=0` alors `pgcd(a,b)=pgcd(b,r)` si `a=b*q+r` avec `0<=r<b`

Dans `Xcas` on a `q=iquo(a,b)` et `r=irem(a,b)`.
Donc le pgcd de a et b est le dernier reste non nul dans les divisions successives de a par b, puis de b par r etc....

Voici le programme de l'algorithme d'Euclide en utilisant la commande `irem(a,b)` qui renvoie le reste de la division euclidienne de l'entier `a` par l'entier `b`.

```
//algo d'Euclide
//a et b sont des entiers positifs ou nuls
pgcd(a,b):={
  local r;
```

```
  tantque b!=0 faire
    r:=irem(a,b);
    a:=b;
    b:=r;
  ftantque;
  retourne a;
}:;
```

Si a est plus petit que b, la première itération de la boucle échange a et b, il n'est donc pas nécessaire de faire un test avant la boucle.

Sans utiliser la commande irem(a,b), on utilise l'instruction :

```
tantque a>=b faire a :=a-b ftantque
```

qui met dans a le reste de la division euclidienne de l'entier a par l'entier b. On tape :

```
//le pgcd avec l'algo d'Euclide sans utiliser irem
pgcde(a,b):={
  tantque b!=0 faire
    tantque a>=b faire a:=a-b;ftantque;
    a,b:=b,a;
  ftantque;
  retourne a;
}:;
```

On peut aussi écrire :

```
//le pgcd avec l'algorithme des differences
pgcdd(a,b):={
  si b==0 alors retourne a fsi;
  si a==0 alors retourne b fsi;
  repeter
    tantque a>b faire a:=a-b;ftantque;
    tantque b>a faire b:=b-a;ftantque;
  jusqua b==a;
  retourne a;
}:;
```

6.2 Pgcd et récursivité

On a :
pgcd(a,0)=a et
pgcd(a,b)=pgcd(b,r) si a=b*q+r avec 0<=r<b
Dans Xcas on a r=irem(a,b) et q=iquo(a,b).
On tape :

```
//le pgcd avec l'algorithme recursif
pgcdr(a,b):={
  si b==0 alors retourne a; fsi;
  retourne pgcdr(b,irem(a,b));
}:;
```

On tape :
`pgcd(32,424),pgcde(32,424),pgcdd(32,424),pgcdr(32,424)`
On obtient :
`8,8,8,8`

7 Le crible d'Eratosthene

7.1 Description

Pour trouver les nombres premiers inférieurs ou égaux à N :

1. On écrit les nombres de 2 à N dans une liste.
2. On met 2 dans la case P .
3. Si $P \times P \leq N$ il faut traiter les éléments de P à N : on barre tous les multiples de P à partir de $P \times P$.
4. On augmente P de 1.
   Si $P \times P$ est strictement supérieur à N, on arrête
5. On met le plus petit élément non barré de la liste dans la case P. On reprend à l'étape 3

7.2 Écriture de l'algorithme avec Xcas

```
//renvoie la liste des nombres premiers<=n selon eratosthene
crible(n):={
  local tab,prem,p,j;
  //on ecrit dans trab la liste des entiers de 0 a n
  tab:=makelist(x->x,0,n);
  //1 n'est pas premier donc on met 0 ds tab[1]
  tab[1]:=0;
  p:=2;
  //on barre les multiples de p qui sont compris entre p*p et n
  //en mettant 0 dans la case correspondante
  tantque (p*p<=n) faire
    pour j de p*p jusque n pas p faire
      tab[j]:=0;
    fpour;
    p:=p+1;
    //on cherche le premier element non barre dans tab
    tantque ((p*p<=n) et (tab[p]==0)) faire
      p:=p+1;
    ftantque;
  ftantque;
  //on remplit la liste prem avec les elements non nuls de tab
  prem:=[];
  pour j de 2 jusque n faire
    si (tab[j]!=0) alors
      prem:=append(prem,j);
    fsi
```

```
  fpour
  retourne(prem);
}:;
```

On tape :
`crible(64)`
On obtient :
`[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61]`

8 Changement de base

8.1 Écriture en base b d'un nombre entier avec 2<=b<=10 à l'aide d'une liste

On écrit un nombre n en base b avec des “chiffres” $b_0, ..., b_s$. Ces chiffres seront représentés par des entiers tels que $1 \leq b_0 < b, 0 \leq b_1 < b, ..., 0 \leq b_s < b$:

$$n = b_0 b^s + b_1 b^{s-1} + ...b_{s-1} b + b_s$$

On les stockera dans une liste de nombres entiers B. Avec `Xcas` on a pour $j = 0..s$, b_j =`B[j]`.

On remarque que :
`irem(n,b)`=b_s et
`iquo(n,b)`=$q = b_0 b^{s-1} + b_1 b^{s-2} + ...b_{s-1}$
D'où `irem(q,b)`=b_{s-1} etc ...
On va donc faire une boucle qui calculera `irem(n,b)` puis qui recommencera avec `n :=iquo(n,b)`.

Il faut faire attention pour générer la liste car on obtient en premier son dernier élement, donc il faudra mettre à chaque ètape l'éément suivant devant ceux que l'on a déjà obtenu....mais heureusement dans `Xcas`, il y a la commande `prepend(L,a)` qui renvoie une liste où lélément `a` est mis au début de la liste `L`.
On tape :

```
//ecriture en base b avec une liste B d'entiers de 0..b-1
dix2B(n,b):={
  local B;
  si n==0 alors retourne [0] fsi;
  B:=[];
  tantque n>0 faire
    B:=prepend(B,irem(n,b));
    n:=iquo(n,b);
  ftantque;
  retourne B;
}:;
```

On tape :
`dix2B(139,7)`
On obtient :
`[2,5,6]`

et on a bien : $6 + 5 * 7 + 2 * 49 = 139$
Remarque
On tape :
`dix2B(139,10)`
On obtient :
`[1,3,9]`
On remarque que `dix2B` peut servir à transformer un nombre entier en la liste des nombres servant à son écriture.

8.2 Écriture d'un nombre entier lorsqu'on donne la liste de ses coefficients en base b avec 2<=b<=10

On entre $B = [b_0, b_1...b_s]$ avec pour $j = 0..s$ $0 <= b_j < b$ et on veut trouver n tel que : $n = b_0b^s + b_1b^{s-1} + ...b_{s-1}b + b_s$.
Remarque
n est la valeur du polynôme $P(x) = b_0x^s + b_1x^{s-1} + ...b_{s-1}x + b_s$ en $x = b$.
On remarque, par exemple pour $s = 4$ que :
$n = b_0b^4 + b_1b^3 + b_2b^2 + b_3b + b_4 = ((((0 * b + b_0)b + b_1)b + b_2)b + b_3)b + b_4$
On initialise `n` à 0 et on calcule `n` en lui ajoutant `B[0]`, puis en le multipliant par `b`, puis, on lui ajoute `B[1]` etc...
On tape :

```
//n est ecrit en base b avec une liste
//pour calculer n en base 10 on retrouve l'algo de Horner
B2dix(B,b):={
  local s,j,n;
  s:=dim(B)-1;
  n:=0;
  pour j de 0 jusque s faire
    //si B[j]>=b alors retourne "erreur" fsi;
    n:=n*b;
    n:=n+B[j];
  fpour;
  retourne n;
}:;
```

On tape :
`B2dix([2,5,6],7)`
On obtient :
`139`
et on a bien : $6 + 5 * 7 + 2 * 49 = 139$
Remarque
On tape :
`B2dix([1,3,9],10)`
On obtient :
`139`
`B2dix([1,3,29],10)`
On obtient :
`159`

On remarque que `B2dix` peut servir à transformer une liste de nombres considérés commes les coefficients d'un polynôme en un nombre entier qui est la valeur du polynôme pour $x =$ le 2-ième argument. C'est pourquoi nous commentons dans `B2dix` le test :
`//si B[j]>=b alors retourne "erreur" fsi;`
Pour savoir si la liste `B` repésente bien une écriture en base `b`, on peut écrire :

```
estbienecritl(B,b):={
  local j,s;
  s:=dim(B)-1;
  pour j de 0 jusque s faire
    si B[j] >= b alors retourne 0; fsi;
  fpour;
  retourne 1;
}:;
```

8.3 Écriture en base b d'un nombre entier avec 2<=b<=10 à l'aide d'une chaîne de caractères

On tape :

```
// ecriture de n en base b dans une chaine
dix2bc(n,b):={
  local C;
  C:="";
  tantque n>0 faire
    C:=irem(n,b)+C;
    n:=iquo(n,b);
  ftantque;
  retourne C;
}:;
```

On tape :
`dix2bc(139,7)`
On obtient :
`"256"`

8.4 Écriture d'un nombre entier lorsqu'on donne ses chiffres en base b avec 2<=b<=10 sous la forme d'une chaîne de caractères

Pour savoir si la chaîne `B` repésente bien une écriture en base `b`, on peut écrire :

```
estbienecritc(B,b):={
  local j,s;
  s:=dim(B)-1;
  pour j de 0 jusque s faire
    si expr(B[j]) >= b alors retourne 0; fsi;
  fpour;
  retourne 1;
}:;
```

Pour convertir, on tape :

```
//nc est ecrit en base b avec une chaine
bc2dix(nc,b):={
  local s,j,n;
  s:=dim(nc)-1;
  n:=0;
  pour j de 0 jusque s faire
    // si expr(nc[j])>=b alors retourne "erreur" fsi;
    n:=n*b;
    n:=n+expr(nc[j]);
  fpour;
  retourne n;
}:;
```

On tape :
`bc2dix( "256",7)`
On obtient :
`139`

8.5 Nombres écrits sans 2 en base 3.

8.5.1 L'énoncé

On veut afficher en base dix la suite ordonnée des entiers dont l'écriture en base trois ne comporte que des 0 ou des 1 (pas de 2).

1. Calculer à la main les huit premiers termes de cette suite.
2. Décrire un algorithme qui donne les 128 premiers termes de cette suite.
3. Écrire une fonction qui renvoie la liste des n premiers termes de cette suite.

8.5.2 La correction

1. Voici les 8 premiers termes de cette suite :
   `[0,1,3,4,9,10,12,13]` dont lécriture en base 3 est :
   `[[0],[1],[1,0],[1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]]`
2. Il y a plusieurs algorithmes possibles :
   - On écrit les nombres de 0 à N en base 3 et on met dans la liste réponse ceux qui ne contiennent pas de 2 dans leur écriture en base 3. On s'arrete quand la liste réponse contient n éléments,
   - On écrit les nombres de 0 à n en base 2 et on considère cette écriture comme étant celle d'un nombre écrit en base 3.
   - On regarde comment les termes de la suite se déduisent les uns des autres.
     On peut remarquer que :
     0=3*0, 1=3*0+1, 3=3*1, 4=3*1+1,
     9=3*3, 10=3*3+1, 12=3*4, 13=3*4+1....
   - On regarde comment sont faits les termes de la suite. On peut remarquer que
     - 3=0+3, 4=1+3 donc 3 et 4 ont èté obtenu à partir de 0 et 1 en leur ajoutant 3,

– 9=0+9, 10=1+9, 12=3+9, 13=4+9 donc 9,10,11 et 12 ont èté obtenu à partir de 0,1,3 et 4 en leur ajoutant 9...

Les 8 prochains termes de cette suite seront obtenus en ajoutant 3^3 à chacun des 8 premiers. On obtient ainsi les 16 premiers termes de cette suite. Puis les 16 prochains termes de cette suite seront obtenus en ajoutant 3^4 à chacun des 16 premiers termes etc...

3. On va traduire avec `Xcas` le premier et le dernier algorithme décrit ci-dessus. Voici les fonctions de `Xcas` que l'on va utiliser :
   – La fonction `est_element(L,a)` qui teste si `a` est dans la liste `L` et qui renvoie `0` ou `n+1` si `n` est l'indice de la première occurence de `a` dans `L`,
   – La fonction `convert(n,base,b)` (resp `convert(L,base,b)`) qui convertit un entier `n` en la liste des coefficients en base `b` dans l'ordre croissant (resp qui convertit la liste `L` des coefficients en base `b` dans l'odre croissant en un entier `n`). On peut aussi utiliser les fonctions `dix2B(n,2)` et `B2dix(L,3)` définies précédement mais les calculs seront plus lents.
   – Pour ajouter un nombre `a` à chacun des termes d'une liste `L` on peut utiliser l'addition de `L` et de la liste `[a,a..a]` de la taille de `L` et écrire `L+[a$dim(L)]`

   Voici les programmes correspondants aux algorithmes 1 et 4 décrits précédement à la question 2. (Pour plus de détails voir le manuel d'algorithmique de `Xcas`).

   On tape les fonctions `pasde21(n)` et `pasde24(n)` qui renvoient les `n` premiers termes de la liste demandée. La variable `p` contient à chaque étape la dimension de la liste `L`.

   –

```
pasde21(n):={
 local L,j,J,p;
 L:=[0];
 p:=1;
 j:=1;
 tantque p<n faire
   J:=convert(j,base,3);
   //J:=dix2B(j,3);
   si not(est_element(2,J)) alors
     L:= append(L,j);
     p:=p+1;
   fsi;
   j:=j+1;
 ftantque;
 retourne L;
}
:;
```

   – La variable `j` contient le nombre d'iterations : à chaque étape on a $p = 2^j$ et `puis3j`$= 3^j$. À la fin de la boucle `tantque`, `L` a $2^j = p >= n$ éléments : il faut donc raccourcir la liste `L` (`L :=mid(L,0,n)`)

```
pasde24(n):={
```

```
 local L,j,p,puis3j;
 L:=[0];
 j:=0;
 p:=1;
 puis3j:=1;
 tantque p<n faire
   L:=concat(L,L+[puis3j$p]);
   //L:=concat(L,map(L,x->x+puis3j));
 j:=j+1;
 puis3j:=3*puis3j;
 p:=2*p;
 ftantque;
 L:=mid(L,0,n);
 retourne L;
}
:;
```

Dans le programme ci-dessus, on calcule des termes pour rien... On modifie donc le programme :

```
pasde24b(n):={
 local L,j,p,puis3j;
 L:=[0];
 j:=0;
 p:=1;
 puis3j:=1;
 tantque 2*p<=n faire
   L:=concat(L,L+[puis3j$p]);
   j:=j+1;
   puis3j:=3*puis3j;
   p:=2*p;
 ftantque;
 L:=concat(L,mid(L,0,n-p)+[puis3j$(n-p)]);;
 retourne L;
}:;
```

Le dernier algorithme est le meilleur...

9 Somme, Produit

9.1 Nombre de chiffres d'un entier

On tape pour ne pas utiliser la fonction `ln` :

```
taille(n):={
  local t;
  si n==0 alors retourne 1; fsi;
  t:=0;
  tantque n!=0 faire
    n:=iquo(n,10);
    t:=t+1;
  ftantque;
```

```
  retourne t;
}:;
```

ou encore on peut se servir de `dix2B` et on écrit :
`taille(n) :=dim(dix2B(n,10))`
On tape :
`taille(123456789)`
On obtient :
`9`

9.2 Somme de 2 grands entiers

On va écrire la fonction somme de deux entiers a et b comme on le fait à la main. Pour cela, on va prendre comme paramètre de la fonction `Somme` les listes A et B constituées par les nombres servant à l'écriture de ces deux entiers (on pourra transformer chacun des 2 nombres a et b en ces listes avec `dix2B` et obtenir A et B).
On complète la plus petite liste en mettant des zéros au début de façon à avoir 2 listes de même longueur.
Au début il n'y a pas de retenue donc `ret :=0`. On remplit la liste `Result`, qui contiendra le résultat final, en commençant par le chiffre des unités c'est à dire par la fin de la liste.
On tape :

```
//on entre des listes de nombres compris entre 0 et 9
//on renvoie une liste de nombres compris entre 0 et 9
Somme(A,B):={
  local j,sa,sb,s,c,ret,Result;
  Result:=[];
  sa:=dim(A)-1;
  sb:=dim(B)-1;
  si sa>sb alors
    B:=concat([0$ (sa-sb)],B);s:=sa;
  sinon
    A:=concat([0$ (sb-sa)],A);s:=sb;
  fsi;
  ret:=0;
  pour j de s jusque 0 pas -1 faire
    c:=A[j]+B[j]+ret;
    Result[j]:=irem(c,10);
    ret:=iquo(c,10);
  fpour;
  si ret>0 alors Result:=prepend(Result,ret); fsi;
  retourne Result;
}
:;
```

On tape par exemple :
`A :=dix2B(1234,10);B :=dix2B(99999,10);`
`S :=Somme(A,B);B2dix(S,10)`

On obtient :
`[1,0,1,2,3,3],101233`
On vérifie : $1234 + 99999 = 101233$
Remarque Avec `Xcas` les polynômes peuvent être représenté sous forme symbolique (par exemple $11x^2 + 22x + 33$) ou sous la forme de la liste de ses coefficients que l'on écrit par ordre décroissant (par exemple [11,22,33]). On peut passer d'une représentation à l'autre avec : `symb2poly` ou `poly2symb`.
Par exemple :
`symb2poly(11*x^2+22*x+33)=[11,22,33]` et
`normal(poly2symb([11,22,33]))=11*x^2+22*x+33`.
Si les listes A et B représentent les coefficients de 2 polynômes (coefficients que l'on écrit par ordre décroissant), alors pour faire la somme de ces 2 polynômes, l'algorithme est le même à condition de ne pas gérer les retenues (par exemple la liste [11,22,33] représente le polynôme $11x^2 + 22x + 33$).
On tape :

```
//A et B sont les listes des coefficients de
//2 polynomes (ordre decroissant)
//on renvoie la liste A+B
Sommep(A,B):={
  local j,sa,sb,s,c,Result;
  Result:=[];
  sa:=dim(A)-1;
  sb:=dim(B)-1;
  si sa>sb alors
    B:=concat([0$ (sa-sb)],B);s:=sa;
  sinon
    A:=concat([0$ (sb-sa)],A);s:=sb;
  fsi;
  ret:=0;
  pour j de s jusque 0 pas -1 faire
    c:=A[j]+B[j];
    Result[j]:=c;
  fpour;
  retourne Result;
}:;
```

On tape par exemple :
`A :=[12,-3,35];`
`B :=[23,1,23,45];`
`S :=Sommep(A,B)`
On obtient :
`[23,13,20,80]`

9.3 Produit d'un entier par un entier a <10

On va écrire la fonction produit d'un entier n par un entier $a < 10$ comme on le fait à la main. Pour cela, on va prendre comme paramètre de la fonction `Prodan` le nombre a et la liste N constituée par les nombres servant à l'écriture de l'entier

n (on pourra transformer le nombres n en cette liste avec `dix2B` et obtenir N).
On tape :

```
//on entre un entier a<10 et une liste N de nombres
//compris entre 0 et 9
//on renvoie une liste de nombres compris entre 0 et 9
Prodan(a,N):={
  local j,sn,c,ret,Result;
  si a>=10 alors retourne "erreur"; fsi;
  Result:=[];
  sn:=dim(N)-1;
  ret:=0;
  pour j de sn jusque 0 pas -1 faire
    c:=a*N[j]+ret;
    Result[j]:=irem(c,10);
    ret:=iquo(c,10);
  fpour;
  si ret>0 alors Result:=prepend(Result,ret); fsi;
  retourne Result;
}
:;
```

On tape par exemple :
`a :=4 ;N :=dix2B(199999,10) ;`
`Pa :=Prodan(a,N) ;B2dix(Pa,10)`
On obtient :
`[7,9,9,9,9,6],799996`
On vérifie : $4 * 199999 = 799996$
Remarque Si la liste P représente les coefficients d'un polynôme (coefficients que l'on écrit par ordre décroissant), alors pour faire la produit de ce polynôme par un réel a, l'algorithme est le même à condition de ne pas gérer les retenues.
On tape :

```
//on entre un entier a et la liste P des coefficients d'un
//polynome (ordre decroissant)
//on renvoie la liste a*P
Prodanp(a,P):={
  local j,sn,c,Result;
  Result:=[];
  sn:=dim(P)-1;
  pour j de sn jusque 0 pas -1 faire
    c:=a*P[j];
    Result[j]:=c;
  fpour;
  retourne Result;
}:;
```

On tape par exemple :
`a :=11 ;`
`P :=[23,1,23,45] ;`

Pa :=Prodanp(a,P)
On obtient :
[253,11,253,495]

9.4 Produit de 2 entiers

On va écrire la fonction produit de deux entiers comme on le fait à la main. Pour cela on va prendre comme paramètre de la fonction `Produit` les listes A et B constituées par les nombres servant à l'écriture de ces deux entiers (on pourra transformer chacun des 2 nombres a et b en ces listes avec `dix2B` et obtenir A et B). On se servira des deux fonctions précédentes.
On tape :

```
//on entre des listes de nombres compris entre 0 et 9
//on renvoie une liste de nombres compris entre 0 et 9
Produit(A,B):={
  local j,sa,sb,C,Result;
  sb:=dim(B)-1;
  Result:=[0$sb];
  pour j de sb jusque 0 pas -1 faire
    C:=Prodan(B[j],A);
    C:=concat(C,[0$(sb-j)]);
    Result:=Somme(Result,C);
  fpour;
  retourne Result;
}:;
```

On tape par exemple :
A :=dix2B(1234,10) ;B :=dix2B(9999,10) ;
P :=Produit(A,B) ;B2dix(P,10)
On obtient :
[1,2,3,3,8,7,6,6],12338766
On vérifie : $1234 * 9999 = 12338766$
Remarque Si les listes A et B représentent les coefficients de 2 polynômes (coefficients que l'on écrit par ordre décroissant), alors pour faire le produit de ces 2 polynômes, l'algorithme est le même à condition de ne pas gérer les retenues.
On tape :

```
//A et B sont les listes des coefficients de
//2 polynomes (ordre decroissant)
//on renvoie la liste qui represente le produit des 2 polynomes
Produitp(A,B):={
  local j,sa,sb,C,Result;
  sb:=dim(B)-1;
  Result:=[0$sb];
  pour j de sb jusque 0 pas -1 faire
    C:=Prodanp(B[j],A);
    C:=concat(C,[0$(sb-j)]);
    Result:=Sommep(Result,C);
```

```
  fpour;
  retourne Result;
}:;
```

On tape par exemple :

```
A :=[12,-3,35];
B :=[23,1,23,45];
P :=Produitp(A,B)
```

On obtient :

```
[276,-57,1078,506,670,1575]
```

On vérifie et on tape :

```
normal(poly2symb(A)*poly2symb(B))
```

On obtient :

```
276*x^5-57*x^4+1078*x^3+506*x^2+670*x+1575
```

10 Jeu de recherche un nombre

10.1 Le jeu

10.2 C'est vous qui jouez

L'ordinateur choisi au hasard un nombre `n` entre 1 et 1000 et vous devez le trouver. Chaque fois que vous proposez une réponse `rep`. l'ordinateur vous dit si le nombre est plus grand ou plus petit que votre réponse ou bien il vous dit "Bravo" en vous donnant votre nombre d'essais.
On tape :

```
je_trouve():={
  local n,rep,j;
  n:=rand(1000)+1;
  saisir("trouvez un nombre entre 1 et 1000",rep);
  j:=1;
  tantque n!=rep faire
    j:=j+1;
    si rep<n alors
      saisir("c'est plus grand que "+rep+", essai "+j,rep);
    sinon
      si rep>n  alors
        saisir("c'est plus petit que "+rep+", essai "+j,rep);
      fsi;
    fsi;
  ftantque;
  retourne "Bravo "+n+" trouve en "+j+" fois";
}:;
```

Pour jouer on tape : `je_trouve()`

10.3 C'est l'ordinateur qui joue

Vous devez choisir un nombre n : on le fait avec `choisir_nombre`. Quelque soit le nombre n, dans le programme `dichot` l'ordinateur choisit le milieu de

l'intervalle dans lequel se trouve ce nombre. Si l'ordinateur sait que le nombre choisi est entre `m` et `M`, l'ordinateur choisit `floor((m+M)/2)` car il doit proposer un nombre entier et la réponse proposée par l'ordinateur est stockée dans `rep`.
Vous devez répondre :

- "+" si votre nombre est plus grand que le nombre proposé,
- "-" si votre nombre est plus petit que le nombre proposé,
- "=" si votre nombre est égal au nombre proposé.

Votre réponse est mise dans la variable `q` : Si vous répondez n'mporte quoi on vous redemande une entée correcte : c'est ce que fait `repeter`. La variable `j` compte le nombre d'essai . On tape :

```
choisir_nombre():={
local n;
saisir("Choisissez un nombre entre 1 et 1000",n);
afficher("l'ordinateur doit trouver ce nombre en vous interrogeant");
retourne n;
}:;
dichot():={
  local j,m,M,rep,q,n;
  m:=1; M:=1000;
  pour j de 1 jusque 11 faire
    rep:=floor((m+M)/2);
    repeter
      saisir_chaine "essai "+j+ ": "+rep+" est-ce +, - ou =",q;
    jusqua q=="+" ou q=="-" ou q=="=";
    si q=="=" alors
      afficher(rep+" trouve en "+j+" essais"); retourne rep;
    fsi;
    si q=="+" alors
      m:=rep+1;
    sinon
      M:=rep-1;
    fsi;
    si m==M alors
      afficher(m+" trouve en "+j+" essais"); retourne m;
    fsi;
  fpour;
}:;
jeu_ordi():={
  local n,rep;
  n:=choisir_nombre();
  rep:=dichot();
  afficher("j'avais bien choisi :"+n);
  retourne rep;
}:;
```

On tape :
`jeu_ordi()`

10.4 Solution approchée de $f(x) = 0$ sur $[a; b]$

On suppose que f est continue et que $f(a) * f(b) < 0$. Donc f s'annule sur $[a; b]$. On va procéder par dichotomie : on partage l'intervalle $[a; b]$ en deux :

$$[a, b] = [a; m = (a + b)/2] \cup [m = (a + b)/2; b]$$

et on regarde si $f(a) * f(m) < 0$ si c'est le cas on cherche la solution dans $[a; m]$ sinon on cherche la solution dans $[m; b]$. On recommence autant de fois qu'il faut pour avoir la précision eps désirée.
On tape :

```
Zero(f,a,b,eps):={
  local m;
  a:=evalf(a); b:=evalf(b);
  si f(a)==0 alors retourne a fsi;
  si f(b)==0 alors retourne b fsi;
  si f(a)*f(b)>0 alors retourne "erreur" fsi;
  tantque (abs(a-b)>eps) faire
    m:=(a+b)/2;
    si (f(a)*f(m)<0) alors
      b:=m;
    sinon
      a:=m;
    fsi;
  ftantque;
  si a<b alors retourne a,b; sinon retourne b,a; fsi;
}:;
```

On tape :
`Zero(x->x^2-2,0,2,1e-10)`
On obtient :
`1.41421356233,1.41421356238`

11 Les carrés emboités

On veut tracer n carrés : un carré direct de côté $AB = d$, puis les carrés directs de côtés AB_j lorsque les B_j définissent une subdivision de AB en n parties égales ($AB_1 = d/n = B_1B_2 = ... = B_{n-1}B$).

11.1 Les carrés emboités et la tortue

11.1.1 Les carrés emboités (itératif)

On tape dans un éditeur de programmes :

```
Carres1(d,n):={
  local j,k;
  k:=d/n;
  pour j de 1 jusque n faire
```

```
    repete(4,avance d,tourne_gauche 90);
    d:=d-k;
  fpour;
}:;
```

11.1.2 Les carrés emboités (récursif)

La figure `Carrer1(d,n)` est composée du grand carré et de `Carrer1(d-d/n,n)`
On tape :

```
Carrer1(d,n):={
  si n<0 alors retourne 1;fsi;
  repete(4,avance d,tourne_gauche 90);
  d:=d-d/n;
  Carrer1(d,n-1);
}
:;
```

On ouvre un écran de géométrie tortue et on tape avec `Alt+d` :

```
efface;
Carres1(100,6);
pas_de_cote -120;
Carrer1(100,6);
```

11.2 Les carrés emboités en géométrie

11.2.1 Les carrés emboités (itératif)

On tape :

```
carres(A,B,n):={
  local j,L,k;
  L:=NULL;
  k:=(B-A)/n;
  pour j de 1 jusque n faire
    B:=translation(k*j,A);
    L:=L,carre(A,B);
  fpour;
  retourne L;
}:;
```

Puis on tape dans une ligne d'entrée :
`carres(point(0),point(100),6)`
et on voit le dessin dans l'écran réponse.

11.2.2 Les carrés emboités (récursif)

On tape dans un éditeur de programmes :

```
carrer(A,B,n):={
  si n<0 alors retourne 1;fsi;
```

```
  carre(A,B);
  B:=translation((B-A)*(n-1)/n,A);
  retourne carrer(A,B,n-1);
}:;
```

Puis on tape :
`DispG(); carrer(point(0),point(100),5)`
et on voit le dessin dans l'écran `DispG` car c'est seulement ce qui est retourné par la fonction qui est dessiné dans la réponse, alors que toutes les objets graphiques intermédiaires sont affichés dans l'écran `DispG`.

Ou bien on tape dans un éditeur de programmes :

```
carrerL(A,B,n,L):={
  si n<0 alors retourne L;fsi;
  L:=[L,carre(A,B)];
  B:=translation((B-A)*(n-1)/n,A);
  retourne carrerL(A,B,n-1,L);
}:;
```

Puis on tape dans une ligne d'entrée :
`carrerL(point(0),point(100),6,[])`
et on voit le dessin dans la réponse.

Remarque pour les utilisateurs avancés
On peut améliorer le programme précédent en utilisant une seule liste que l'on modifie en place (avec l'opérateur `=<`) afin de ne pas recopier la liste `L` à chaque affectation par `:=` (un peu comme avec un passage d'arguments par référence). Attention, toutefois, cela nécessite de faire une copie de la liste vide initiale par `copy` sinon c'est la liste du programme lui-même qui sera modifiée et ne sera donc plus initialisée à une liste vide.

```
carreenplace(A,B,n,L):={
  si n<0 alors retourne L;fsi;
  L[dim(L)] =< carre(A,B);
  B:=translation((B-A)*(n-1)/n,A);
  retourne carreenplace(A,B,n-1,L);
}:;

appel_carre(A,B,n):={
  local L;
  L:=copy([]);
  carreenplace(A,B,n,L);
  retourne L;
}:;
```

12 D'autres carrés emboités

On veut tracer n carrés : on trace un carré direct de côté $AB = d$, puis le carré de sommets les milieux des côtés du carré dessiné précédemment...

12.1 Avec la tortue

12.1.1 Les carrés (itératif)

On tape :

```
Carres21(d,n):={
  local j;
  pour j de 1 jusque n faire
    repete 4,avance d,tourne_gauche 90;
    avance d/2;
    tourne_gauche 45;
    d:=d/sqrt(2.);
  fpour;
  pour j de 1 jusque n faire
    d:=d*sqrt(2.);
    tourne_droite 45;
    avance -d/2;
  fpour;
}:;
```

On tape si on ne veut pas repasser par les mêmes traits :

```
Carres22(d,n):={
  local c,j,k;
  c:=d;
  j:=1;
  tantque j!=n+1 faire
    avance c/2;
    tourne_gauche 45;
    c:=c/sqrt(2.);
    j:=j+1;
  ftantque;
  tantque j!=1 faire
    tourne_droite 45;
    c:=c*sqrt(2.);
    avance c/2;
    k:=1;
    tantque k!=4 faire
      tourne_gauche 90;
      avance c;
      k:=k+1;
    ftantque;
    tourne_gauche 90;
    j:=j-1;
  ftantque;
}:;
```

12.1.2 Les carrés (récursif)

On tape :

```
Carrer21(d,n):={
  si n<=0 alors retourne 1; fsi;
  repete 4,avance d,tourne_gauche 90;
  avance d/2;
  tourne_gauche 45;
  Carrer21(d/sqrt(2.),n-1);
  tourne_droite 45;
  recule d/2;
}:;
```

On tape si on ne veut pas repasser par les mêmes traits :

```
Carrer22(d,n):={
  si n>0 alors
    avance d/2;
    tourne_gauche 45;
    Carrer22(d/sqrt(2.),n-1);
    tourne_droite 45;
    avance d/2;
    tourne_gauche 90;
    repete 3,avance d,tourne_gauche 90;
  fsi;
}:;
```

On ouvre un écran de géométrie tortue et on tape avec `Alt+d` :

```
efface;
Carres21(100,6);
saute -120;
Carres22(100,6);
pas_de_cote -120;
Carrer21(100,6);saute 120;
Carrer22(100,6);
```

12.2 Les carrés en géométrie

12.2.1 Les carrés (itératif)

On tape :

```
carres2(A,B,n):={
  local j,L,C;
  L:=NULL;
  pour j de 1 jusque n faire
    L:=L,carre(A,B,C);
    A:=point((A+B)/2);
    B:=point((C+B)/2);
  fpour;
  retourne L;
}:;
```

Puis on tape dans une ligne de commande :
`carres2(point(0),point(100),6)`
et on voit le dessin dans l'écran réponse.

12.2.2 Les carrés (récursif)

On tape :

```
carrer2(A,B,n):={
  local C;
  si n==1 alors retourne carre(A,B) fsi;
  carre(A,B,C);
  A:=point((A+B)/2);
  B:=point((C+B)/2);
  retourne carrer2(A,B,n-1);
}:;
```

Puis on tape :
`carrer2(point(0),point(100),6)`
et on voit le dessin dans l'écran `DispG`.

Ou bien on tape :

```
carrerL2(A,B,n,L):={
  local C;
  si n==0 alors retourne L fsi;
  L:=[L,carre(A,B,C)];
  A:=point((A+B)/2);
  B:=point((C+B)/2);
  retourne carrerL2(A,B,n-1,L);
}:;
```

On tape alors dans une ligne de commande :
`carrerL2(point(0),point(100),6,[])`
et on voit le dessin dans l'écran réponse.